Safe Learning for Near-Optimal Scheduling

QEST'21

Damien Busatto-Gaston, Université libre de Bruxelles
Debraj Chakraborty, Université libre de Bruxelles
Shibashis Guha, Tata Institute of Fundamental Research
Guillermo A. Pérez, University of Antwerp
Jean-François Raskin, Université libre de Bruxelles

August 25, 2021

Scheduling hard and soft tasks

- Task system: Set of tasks partitioned into hard and soft tasks H and F.
- Each task generates instances called jobs.
- Tasks are preemptible: the scheduler can stall one job, and continue another job

Scheduling hard and soft tasks

- Task system: Set of tasks partitioned into hard and soft tasks H and F.
- Each task generates instances called jobs.
- Tasks are preemptible: the scheduler can stall one job, and continue another job

Tasks

Tasks are tuples (C, D, A) such that

- $D \in \mathbb{N}$ is the (relative) deadline of all jobs generated by the task
- $C : \{1, 2, \ldots, D\} \to [0, 1]$ is a discrete probability distribution over possible job-computation times,
- $A : \{D, D + 1, \ldots\} \to [0, 1]$ is a distribution over finitely many possible inter-arrival times.

Scheduling hard and soft tasks

- Task system: Set of tasks partitioned into hard and soft tasks H and F.
- Each task generates instances called jobs.
- Tasks are preemptible: the scheduler can stall one job, and continue another job

Tasks

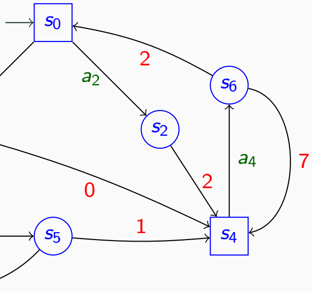
Tasks are tuples (C, D, A) such that

- $D \in \mathbb{N}$ is the (relative) deadline of all jobs generated by the task
- $C : \{1, 2, \ldots, D\} \to [0, 1]$ is a discrete probability distribution over possible job-computation times,
- $A : \{D, D + 1, \ldots\} \to [0, 1]$ is a distribution over finitely many possible inter-arrival times.

- Hard tasks should never miss a deadline
- soft tasks have an associated cost $c \in \mathbb{Q}, c \geq 0$.

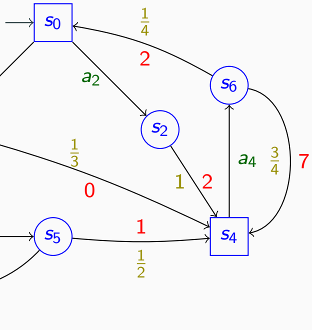
Markov decision process

We model a task system as a Markov decision processes.



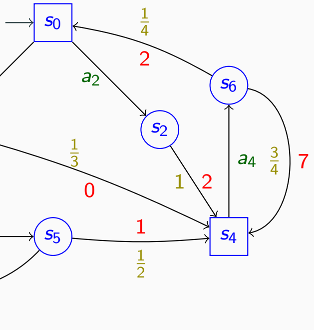
Markov decision process

We model a task system as a Markov decision processes.



Markov decision process

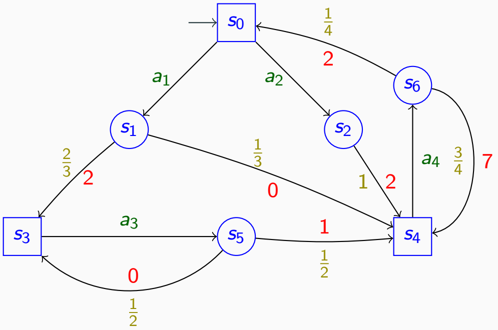
We model a task system as a Markov decision processes.



- Play in the MDP: s_0 Total cost: 0

Markov decision process

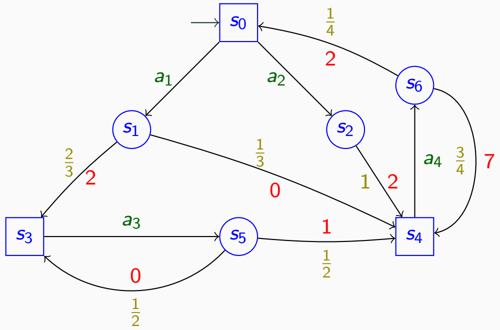
We model a task system as a Markov decision processes.



- Play in the MDP: $s_0 \xrightarrow{a_1} s_1 \xrightarrow{2} s_3$ Total cost: 2

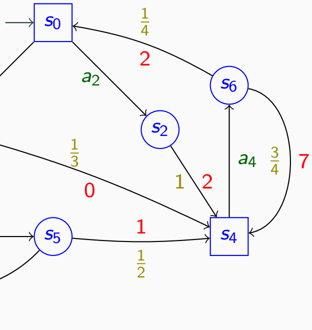
Markov decision process

We model a task system as a Markov decision processes.



- Play in the MDP: $s_0 \xrightarrow{a_1} s_1 \xrightarrow{2} s_3 \xrightarrow{a_3} s_5 \xrightarrow{1} s_4$ Total cost: 3

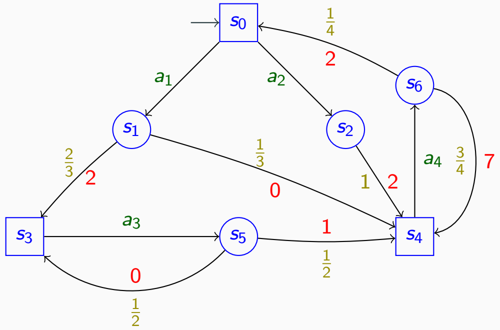
We model a task system as a Markov decision processes.



- Play in the MDP: $s_0 \xrightarrow{a_1} s_1 \xrightarrow{2} s_3 \xrightarrow{a_3} s_5 \xrightarrow{1} s_4 \ldots$ Total cost: 3

Markov decision process

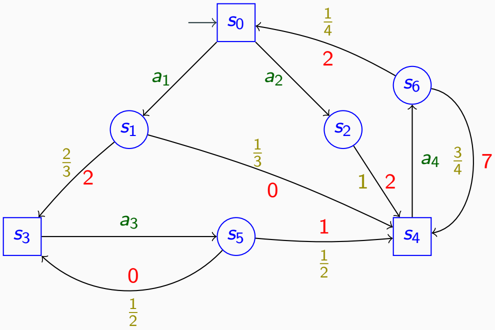
We model a task system as a Markov decision processes.



- Play in the MDP: $s_0 \xrightarrow{a_1} s_1 \xrightarrow{2} s_3 \xrightarrow{a_3} s_5 \xrightarrow{1} s_4 \ldots$ Total cost: 3
- Mean cost: $\lim_{n \to \infty} \frac{1}{n} \sum_{i=0}^{n-1} cost_i$, where $cost_i$ is the cost at i^{th} step

Markov decision process

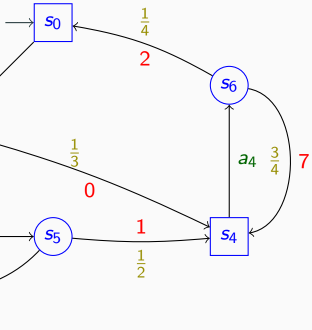
We model a task system as a Markov decision processes.



- Strategy $\sigma : Path_\Box \to Actions$

Markov decision process

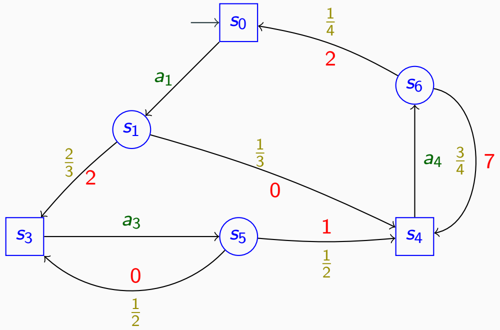
We model a task system as a Markov decision processes.



- Strategy $\sigma : Path_{\square} \to Actions$ creates a Markov chain $\Gamma[\sigma]$

Markov decision process

We model a task system as a Markov decision processes.



- Strategy $\sigma : Path_{\square} \to Actions$ creates a Markov chain $\Gamma[\sigma]$
- Expected mean cost of strategy σ: $\mathbb{E}\left[MeanCost_{\sigma}\right]$

MDP for scheduling problem

Consider the following task system with two tasks:

Task type	C	D	A	Cost
Hard	1	2	3	*n/a*
Soft	$[1 : 0.4, 2 : 0.6]$	2	3	10

MDP for scheduling problem

Consider the following task system with two tasks:

Task type	C	D	A	Cost
Hard	1	2	3	*n/a*
Soft	[1 : 0.4, 2 : 0.6]	2	3	10

States

The states of the MDP contain the following information for each tasks:

- the remaining time $\hat{D} \leq D$ to deadline,
- a distribution $\hat{C} : \{1, 2, \ldots, \hat{D}\} \rightarrow [0, 1]$ over the possible remaining computation times,
- a distribution $\hat{A} : \{\hat{D}, \hat{D} + 1, \ldots\} \rightarrow [0, 1]$ over the possible times before next arrival of a job of that task.

MDP for scheduling problem

Consider the following task system with two tasks:

Task type	C	D	A	Cost
Hard	1	2	3	*n/a*
Soft	$[1:0.4, 2:0.6]$	2	3	10

States

The states of the MDP contain the following information for each tasks:

- the remaining time $\hat{D} \leq D$ to deadline,
- a distribution $\hat{C} : \{1, 2, \ldots, \hat{D}\} \rightarrow [0, 1]$ over the possible remaining computation times,
- a distribution $\hat{A} : \{\hat{D}, \hat{D}+1, \ldots\} \rightarrow [0, 1]$ over the possible times before next arrival of a job of that task.

The initial state :

	$\hat{C}$	$\hat{D}$	$\hat{A}$
H	1	2	3
S	[1:.4,2:.6]	2	3

MDP for scheduling problem

Actions

The action of the MDP:
For Scheduler $\square$:

- Either chooses an active task and gives it one CPU time unit for execution,
- Stays idle (ε)

For Task Generator $\bigcirc$:

- Stays idle (ε),
- Finish the current job (*fin*),
- Submit a new job (*sub*),
- Kill a soft task job and submit a new one (*kill&sub*)

MDP for scheduling problem

Actions

The action of the MDP:
For Scheduler □:

- Either chooses an active task and gives it one CPU time unit for execution,
- Stays idle (ε)

For Task Generator ○:

- Stays idle (ε),
- Finish the current job (*fin*),
- Submit a new job (*sub*),
- Kill a soft task job and submit a new one (*kill&sub*)

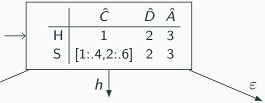
We update the next states accordingly.

MDP for scheduling problem

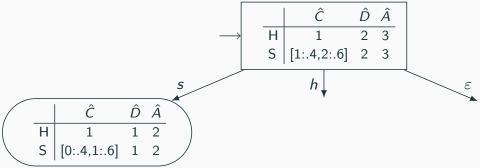
$\longrightarrow$

	$\hat{C}$	$\hat{D}$	$\hat{A}$
H	1	2	3
S	[1:.4,2:.6]	2	3

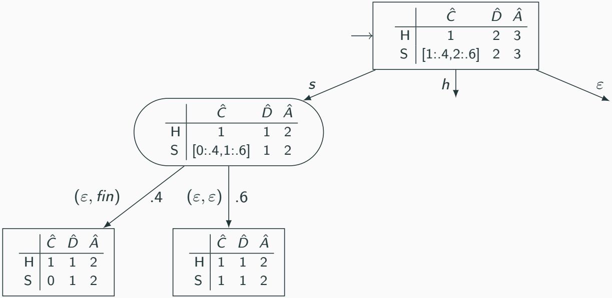
MDP for scheduling problem



MDP for scheduling problem

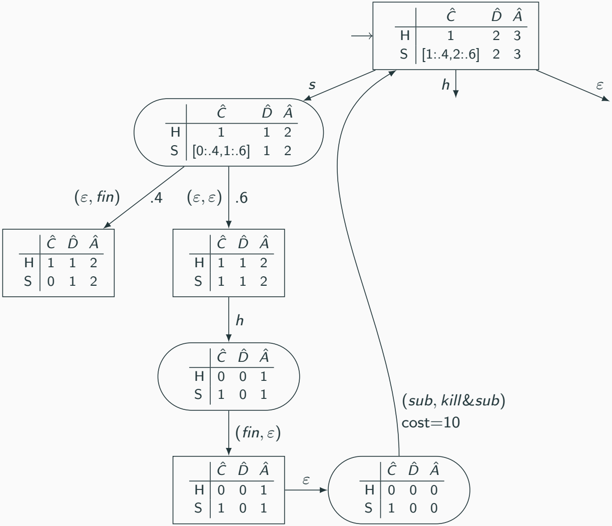


MDP for scheduling problem

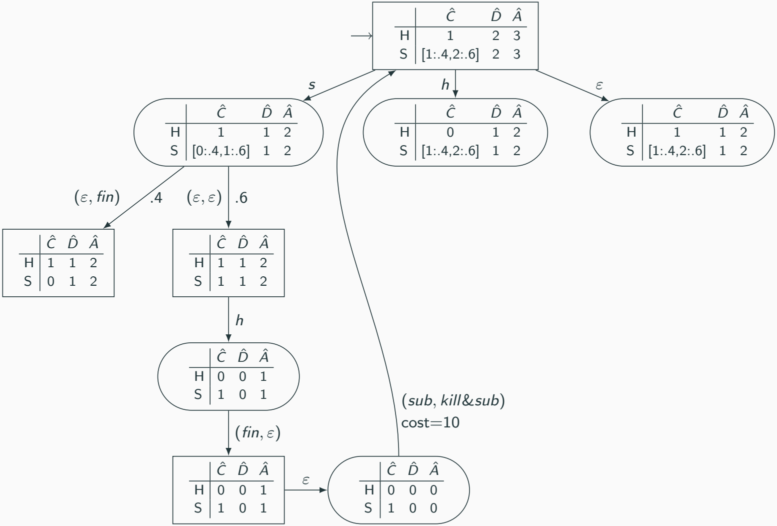


MDP for scheduling problem

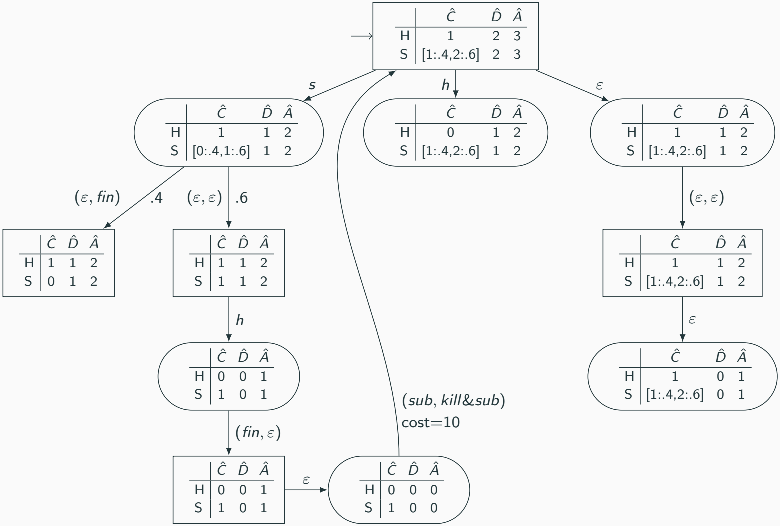
MDP for scheduling problem



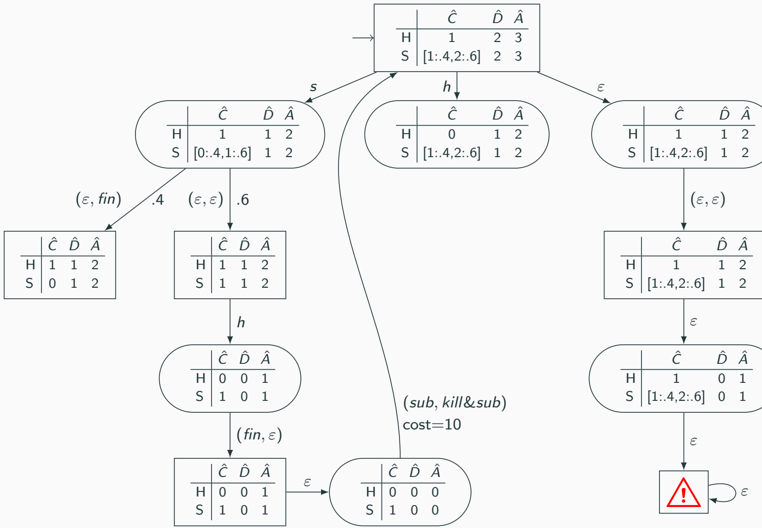
MDP for scheduling problem



MDP for scheduling problem



MDP for scheduling problem



MDP for scheduling problem

Objective

Find a stratgy for scheduler that

- avoids the state ⚠ (denoted some hard task missing deadline),
- minimizes the expected mean-cost

MDP for scheduling problem

Objective

Find a stratgy for scheduler that

- avoids the state ⚠ (denoted some hard task missing deadline),
- minimizes the expected mean-cost

- Prune the MDP to obtain safe region: from all vertices scheduler has a strategy ensuring to visit only safe vertices
  - Polynomial time algorithm in the size of the MDP
- Find the strategy that minimizes the expected mean-cost in the safe region
  - Value iteration (Storm)

Model-based learning

Our settings

- Deadlines and the domains of the distributions are known
- Execution and inter-arrival time distributions are not known
- Need to sample to get "ϵ-close" distributions

Model-based learning

Our settings

- Deadlines and the domains of the distributions are known
- Execution and inter-arrival time distributions are not known
- Need to sample to get "ϵ-close" distributions

$p \sim^{\epsilon} q$ means $\forall a, \mid p(a) - q(a) \mid \leq \epsilon$

Model-based learning

Our settings

- Deadlines and the domains of the distributions are known
- Execution and inter-arrival time distributions are not known
- Need to sample to get "ϵ-close" distributions

Guarantees about learning

- Probably approximately correct (PAC): for all $\epsilon, \gamma \in (0, 1)$, can compute an ϵ-close task system, with probability $\geq 1 - \gamma$.
- safely PAC learnable: PAC learnable, and can ensure safety for the hard tasks while computing the approximation.
- (safely) efficiently PAC learnable : (safely) PAC learnable, and can compute the approximation in $PTIME$(size of the task system, $\frac{1}{\epsilon}$, $\frac{1}{\gamma}$)

Model-based learning

How would we learn an unknown distribution p?

- Collect samples according to the distribution p
- Calculate the relative frequency:
  $$r(a) = \frac{\text{number of times } a \text{ is sampled}}{\text{total number of samples}}$$
  for all $a \in Dom(p)$
- $\forall \epsilon, \gamma \in (0,1)$, if number of samples is "big enough", r is ϵ-close to p with probability $\geq 1 - \gamma$

Model-based learning

How would we learn an unknown distribution p?

- Collect samples according to the distribution p
- Calculate the relative frequency:
$$r(a) = \frac{\text{number of times } a \text{ is sampled}}{\text{total number of samples}}$$
for all $a \in Dom(p)$
- $\forall \epsilon, \gamma \in (0, 1)$, if number of samples is "big enough", r is ϵ-close to p with probability $\geq 1 - \gamma$

still polynomial in size of the domain of p, $\frac{1}{\epsilon}$, $\frac{1}{\gamma}$

Model-based learning

How would we learn an unknown distribution p?

- Collect samples according to the distribution p
- Calculate the relative frequency:
$$r(a) = \frac{\text{number of times } a \text{ is sampled}}{\text{total number of samples}}$$
for all $a \in Dom(p)$
- $\forall \epsilon, \gamma \in (0, 1)$, if number of samples is "big enough", r is ϵ-close to p with probability $\geq 1 - \gamma$

:) Efficiently PAC-learnable

Model-based learning

Learning a task system with no hard tasks

for all tasks, repeat:

- Schedule the task when a job of this task is active till we collect enough samples of inter-arrival and computation time.
- Approximate the inter-arrival time and computation time distribution.

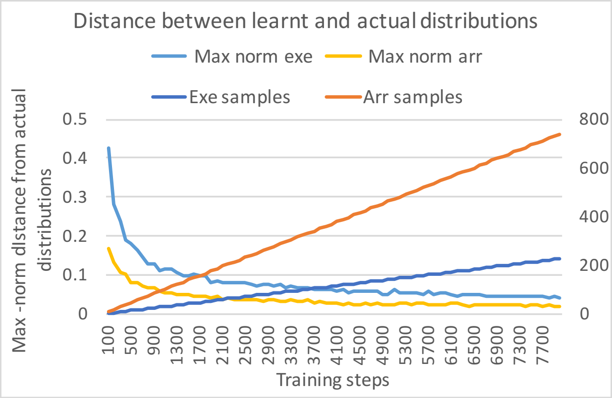
Learning a task system with no hard tasks

for all tasks, repeat:

- Schedule the task when a job of this task is active till we collect enough samples of inter-arrival and computation time.
- Approximate the inter-arrival time and computation time distribution.

:) Efficiently PAC-learnable

Model-based Learning



Learning distributions for a system with 6 soft tasks.

Model-based learning

Learning a task system with hard tasks

- :(Have to follow a safe scheduling strategy to learn safely
- :(May not observe enough samples if we follow a safe strategy
- :(Need a stronger condition on the task system

Model-based learning

Learning a task system with hard tasks

:(Have to follow a safe scheduling strategy to learn safely

:(May not observe enough samples if we follow a safe strategy

:(Need a stronger condition on the task system

Condition: good for sampling

For all soft tasks i, the safe region contains a state v_i where

- a new job of task i enters the system
- there exists a strategy σ_i that safely schedules the hard tasks and under σ_i, this new job is guaranteed to finish before deadline.

Model-based learning

Learning a task system with hard tasks

:(Have to follow a safe scheduling strategy to learn safely

:(May not observe enough samples if we follow a safe strategy

:(Need a stronger condition on the task system

Condition: good for sampling

For all soft tasks i, the safe region contains a state v_i where

- a new job of task i enters the system
- there exists a strategy σ_i that safely schedules the hard tasks and under σ_i, this new job is guaranteed to finish before deadline.

:) Safely PAC-learnable

Learning a task system with hard tasks

- :(Have to follow a safe scheduling strategy to learn safely
- :(May not observe enough samples if we follow a safe strategy
- :(Need a stronger condition on the task system

Condition: good for sampling

For all soft tasks i, the safe region contains a state v_i where

- a new job of task i enters the system
- there exists a strategy σ_i that safely schedules the hard tasks and under σ_i, this new job is guaranteed to finish before deadline.

- :) Safely PAC-learnable
- :(We cannot bound the time needed to get the next sample by a polynomial
- :(Not efficiently PAC-learnable

Learning a task system with hard tasks

More restrictive condition: good for efficient sampling

For all soft tasks, there is a set of scheduler vertices $Safe_i$ in the safe region such that

- from $Safe_i$, there is a strategy, under which all hard tasks and the task i can be safely scheduled
- there is a safe strategy σ_i for the hard tasks such that from any state in the safe region, $Safe_i$ is reachable within $K \in \mathbb{N}$ (polynomial in size of the task system) steps using σ_i.

Learning a task system with hard tasks

More restrictive condition: good for efficient sampling

For all soft tasks, there is a set of scheduler vertices $Safe_i$ in the safe region such that

- from $Safe_i$, there is a strategy, under which all hard tasks and the task i can be safely scheduled
- there is a safe strategy σ_i for the hard tasks such that from any state in the safe region, $Safe_i$ is reachable within $K \in \mathbb{N}$ (polynomial in size of the task system) steps using σ_i.

:) Safely and efficiently PAC-learnable

Example

Consider the following task system:

Task id	Task type	C	D	A	Cost
1	Hard	1	2	4	n/a
2	Soft	$[1 : ?, 2 : ?]$	2	3	10

We do not have a safe schedule that can ensure the soft task never misses a deadline.

- $Safe_2 = \phi$
- 'Good for efficient sampling' condition does not hold

But at time $12n + 6$, $n \geq 0$:

- a new job by the soft task enters the system
- this new job can be scheduled and guaranteed to finish under a safe strategy
- 'Good for sampling' condition holds

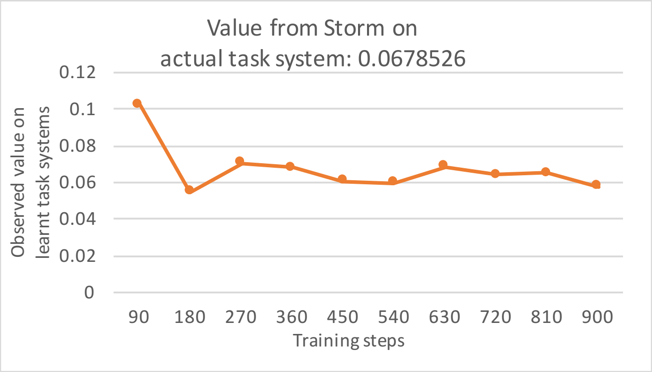
Model-based learning

Using the learnt model

Given a task system Υ, $\beta, \gamma \in (0, 1)$:

- Calculate appropriate ϵ
- Learn a system Υ^M, which is ϵ-close to Υ with probability $\geq 1 - \gamma$
- Compute optimal safe scheduling strategy σ in the MDP corresponding to Υ^M

- σ is a safe strategy in Υ
- With probability $\geq 1 - \gamma$, in Υ, $|\mathbb{E}\left[MeanCost_\sigma\right] - \min_\tau \mathbb{E}\left[MeanCost_\tau\right]| \leq \beta$



Model-based learning for 1 hard, 2 soft tasks

Model-free learning

:(STORM can handle relatively smaller task systems (3-4 tasks $\approx 10^6$ states)

Model-free learning

:(STORM can handle relatively smaller task systems (3-4 tasks $\approx 10^6$ states)

Receding horizon framework

* Fix a horizon H
* At each step, find the best action based on a unfolding tree of depth H
* Use heuristic algorithms like Monte-carlo tree seach
* Prune the tree by adding domain-knowledge using symbolic advice during selection and simulation

Model-free learning

:(Storm can handle relatively smaller task systems (3-4 tasks $\approx 10^6$ states)

Receding horizon framework

- Fix a horizon H
- At each step, find the best action based on a unfolding tree of depth H
- Use heuristic algorithms like Monte-carlo tree seach
- Prune the tree by adding domain-knowledge using symbolic advice during selection and simulation

Deep Q-learning

- Use discount factor close to 1
- Use shielding to restrict actions during learning process so that only safe actions can be used

Model-free learning

Strategies used for advice in MCTS and shielding in deep Q-learning

Earliest deadline first

- Always schedule the jobs by hard tasks with earliest deadline
- If no hard tasks are active, allow jobs by soft tasks

Strategies used for advice in MCTS and shielding in deep Q-learning

Earliest deadline first

- Always schedule the jobs by hard tasks with earliest deadline
- If no hard tasks are active, allow jobs by soft tasks

:) easy to calculate, can be applied to larger systems

:(too restrictive

Strategies used for advice in MCTS and shielding in deep Q-learning

Earliest deadline first

- Always schedule the jobs by hard tasks with earliest deadline
- If no hard tasks are active, allow jobs by soft tasks

:) easy to calculate, can be applied to larger systems

:(too restrictive

Most general safe scheduler

- Allow all safe edges from a scheduler vertex

Model-free learning

Strategies used for advice in MCTS and shielding in deep Q-learning

Earliest deadline first

- Always schedule the jobs by hard tasks with earliest deadline
- If no hard tasks are active, allow jobs by soft tasks

:) easy to calculate, can be applied to larger systems

:(too restrictive

Most general safe scheduler

- Allow all safe edges from a scheduler vertex

:) allows for maximal exploration

:(need to be precomputed (AbsSynth)

Experimental results

Task	MDP size	STORM output	MCTS unsafe	MCTS MGS	MCTS EDF	Deep-Q unsafe	Deep-Q MGS	Deep-Q EDF
4S	10^5	0.38	0.52	NA	NA	0.56	NA	NA
5S	10^6	TimeOut	0	NA	NA	0.13	NA	NA
10S	10^{18}	TimeOut	0	NA	NA	0.96	NA	NA
1H, 2S	10^4	0.07	0.67	0.14	0.28	0.24	0.11	0.22
1H, 3S	10^5	0.28	1.13	0.45	0.49	∞	0.47	0.47
2H, 1S	10^4	0	0.92	0	0.2	∞	0.02	0.3
2H, 5S	10^{10}	TimeOut	3.44	1.93	2.14	∞	2.39	2.48
3H, 6S	10^{14}	TimeOut	4.17	2.88	2.97	∞	3.42	3.47
2H, 10S	10^{22}	TimeOut	0.3	0.03	0.03	∞	1.42	1.6
4H, 12S	10^{30}	TimeOut	2.1	1.2	1.3	∞	2.68	2.87

Comparison of MCTS and reinforcement learning.[1] [2]

[1] ∞ refers to task systems missing deadline for hard tasks

[2] The values reported for both MCTS and Q-learning are obtained as an average cost over 600 steps.

Thank You!